

## ЛЕКЦИЯ 4: Массивы. Указатели.

Двоичный ввод-вывод:  
Двоичный ввод-вывод - запись "шрифта" данных.

- + быстро;
  - + проще хранить большие данные;
  - + можно для чисел с плавающей точкой.
- 16-ричные числа с плавающей точкой:  $0x \dots$   $p \pm exF$

Пример:  $0x1.8 p 1$   $\rightarrow$   $1.2 e^1$   $\rightarrow$   $1.2 \times 10^1$

$$0x1.8 p 1 = 3$$

$$0001.1000 \cdot 2^1$$

$$1.1 \cdot 2^1 - умножение на 2 в двоичной системе = сдвиг вправо на один разряд$$

$$11. \rightarrow 3$$

Пример:  $0x6.8 p -2$

$$0110.1000 \cdot 2^{-2}$$

$$110.1 \cdot \frac{1}{4} \rightarrow 1.625$$

Задание: сколько цифр после точки нужно ввести, если начали всегда  $0x1.0$ , и нужно ввести число типа типа  $float$ :

Ответ: 6 (точность float = 24 бита)

(предикатор  $\%a$ : для негативного числа может быть другое значение)

размерность выводится 6 ( $float, double...$ ) цифр после (.)

массивы.

Одномерный массив.

int  $a[10]$   $\rightarrow$  тип данных называется массивом

размер массива

Обратите внимание к элементу  $\neq$  массиву (>) или вводист мусор, или уронит программу!

Память выделяется структурами:

- Всегда за структуру и за массив  $\Rightarrow UB$
- не всегда за структуру, но  $\Rightarrow UB$  какой-то мусор

UB - undefined behavior

Инициализация массива:

int q[10] = { 1, 2, 3 }

q[0] = 1, q[1] = 2, q[2] = 3, q[4]...q[9] = 0

Если массив не инициализирован, то в нём нули.

При создании массива:

муже

int q[10]

Инициализация - задание параметров (знат-  
ких) в момент создания.

инициализации

int q[10] = { 1, 2, 3 };      int q[10];

q[0] = 1

Задание значения работаем только в момент инициализации.

работаем

int q[10] = { 1, 2, 3 };

не работаем

int q[10];  
q = { 1, 2, 3 }

Массив на один элемент: int q[] = { 0 }

size\_t : количество размеров объектов, массивов от-  
лично необходим.

Сл.: массивы загиваются ровно столько па-  
мяти, сколько нужно для хранения  
одного элемента типа данных. как-то  
размеров. (т.е. отсутствует избыточная  
информация)

sizeof(<имя массива>) = размер массива в char

sizeof(<имя массива>) / sizeof(<имя массива>[0]) =

= кол-во элементов в массиве.

Задание значений при инициализации в  
последовательном порядке

int q[10] = { [2] = 3, [1] = 4 } ;

## Двумерный массив

int  $q[2][3]$ ; | int  $q[2][2][3]$ ; (1)  
 ↑      ↑      ↑  
 y      x      z      y      x      (2)

Двумерный массив: - это непрерывная область памяти фиксированного размера

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]

[0][0], [0][1], [0][2], [1][0],  
 [1][1], [1][2]

представление двумерного массива в памяти

логическое представление двумерного массива

Обращение к памяти одно в отличие от массива массивов.

Си: int  $q[2][3]$

int  $q[\text{const}][3]$

int  $q[2]$

int  $q[x]$

int  $q[\text{const-expr}]$

можно с константами времени компиляции

нельзя с временем компиляции / переменными

Си: VLA (C11) - опциональная поддержка  
 VLA (C99) - обязательная

VLA - возможность создавать массивы помимо НЕ констант времени компиляции

Re: Все локальные переменные на стеке.

? Не стоит создавать массивы на стеке, т.к. переносение адреса ведёт к исчезновению программы.

```
int main()
{
    int q[2][3]; *
```

VLA работает

int w[] \*

```
int main()
{
    static int v[]; *
    int q[2][3];
}
```

VLA не работает

? В ЛР место на стеке с помощью VLA лучше не выделять.

## Варианты использования VLA:

- 1) создание массива - с СИ опционально
- 2) < не допущено на лекции>

Статические и динамические массивы по умолчанию инициализируются 0.

static переменная - динамическая переменная фактически, время жизни

внутри

Си: static int w = argc; - недопустимо.

### Указатели.

#### 1 Указатели \*

Синтаксис символа \*:

- умножение : a \* b
- при создании переменной : "переменная ~~имя~~ указателя" : int \*p;
- унарная \* - обращение сквозь указатель : \*p = 1

Указатель - объект, хранищий адрес другого объекта.

Си: указатели типизированы

int \* p;      int \* p;      int \* p;

float f;

\*p = f;

p = &g[1];

! Синтаксис ! : берёт адрес конкретного объекта под конкретный адрес (не выделяется)

! адрес float не конвертируется автоматически в адрес типа int.

int \* p;

p = &a;

\*p = 1;

Результат: a = 1

Синтаксический указатель содержит тип данных, на объект которого ссылается.

```
int *p; } UB  
*p = 1;
```

```
int a = 2, b;  
const int c = 4;  
int *p;
```

```
p = &c;  
*p = 1
```

! нельзя записывать адрес const объекта.

Указатели часто используются для изменения объектов.

```
const int *p;
```

```
p = &c;      p = &a;  
*p = t;      b = *p;
```

```
const int *p = &a; // инициализация указателя при создании.
```

```
int *const p = &a; // p - константный указатель на объект типа int
```

В этом случае сквозь p можно писать и читать, но записывать нельзя.

\* const p - const относится к p

const \* p - const относится к тому, на что p указывает.

Re: В чем разница?

```
int *p, *x;
```

указатели просто int  
как типы

```
int *p, *x;
```

где указатели

Указатель на указатель

int \*\*z = &p;

указатель объекта типа указатель на int.  
на указатель на объект типа указатель на int)

\*z = &a; // адрес z занесет адрес а в p (p=&a)  
\*\*z = 3; // a = 3.

## 2 Указатель на void (неконкру. указатель):

void \*z;

Невозможно разыменовать, прочитать и переписать.

Указатели на void совместимы с любыми указателями, поэтому:

void \*v;  
v = p; но если const int c=4;, то  
v = z; void \*v; const void \*v;  
v = fp; v=&c; v=&c;  
не работаем работает

ли: верно и в обратную сторону:

void \*v;

p = v;

z = v;

fp = v;

## Взаимодействие массивов и указателей

\*q = 1 // q[0] = 1.

Член массива при исп. указателя = начало этого массива.

Массивы совместимы с указателями.

int p, q[20];

p = q;

p[0] = 2; // q[0] = 2;

p[3] = 4; // q[3] = 4;

q ≠ p;

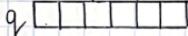
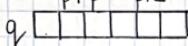
нельзя членов присвоить в один массив

один массив = один указатель.

```
int q[20],  
int *p;  
p = &q[1];  
p[0] = 2; // q[1] = 2  
p[3] = 4; // q[4] = 4
```

## Арифметика указателей

Смещение на  $<\dots>$  объектов вперед / назад:  $p \pm <\dots>$



Не применяется к void.

Примеры исп. арифметики указателей:

$$1) * (p + 2) \equiv p[2]. \quad 2) * (2 + p) \equiv 2[p]$$

$$3) p = \&q[i]$$

$$p[-1] = 2$$

Вычитание указателей возможна, если они указывают на один и тот же объект.

$$p = \&q[1] \quad // \quad p = q + 1$$

$$\left. \begin{array}{l} p - q // 1 \\ q - p // -1 \end{array} \right\} \text{Вычитание расстояний}$$

ptr diff\_t - тип разницы указателей

Умножение, деление и сложение указателей не являются вычислениями.

## Динамическое выделение памяти

int \*p; как-то память для выделения в байтах  
p = malloc(sizeof(int) \* 100); // ручное выделение памяти для динамического выделения памяти

Си: тип  $p = \text{malloc}(\dots)$  - указатель на void.

Си:  $p = \text{malloc}(\dots)$  - компилируется, но в C++ - нет.  
 $\text{free}(p);$

! каждый malloc требует вызова free

```
int *p;
```

```
p = malloc(sizeof(int) * 100);
```

```
if (!p) → error;      обязательная проверка на успех выделения памяти
```

```
free(p);
```

```
int *p = NULL;
```

```
if ()
```

```
{     p = malloc(sizeof(int) * 100);
```

```
}
```

```
free(p);
```

↑ *проверяем с NULL*      } *где раза не вызываем free(p);*

**Чи:** массив *неизвестно* передать. В другую функцию.

void f(int a, int w[10])

~int \*w

```
int main()
```

```
{     int q[10];  
     f(3, q);
```

В другую функцию передаётся указатель на массив, но не сам массив.

Передача по значению

```
void f(int *a, int *w).      без * означает копия объектов
```

w[3] = 1;  
\*a = 2;

```
int main()
```

```
{     int q[10], b = 4;  
     f(&b, q); // q[3] = 1  
   }
```

// b = 2

## многомерная адресация

```
int main()
{
    int w * h;
    int * p = malloc(sizeof(int)*w*h);
    p[y*w + x];
}
```

f(w, h, p); *пунктое построение двумерного массива.*

Примечество в том, что мы получим указатели

```
void f(uint w, uint h, int *z)
```

```
{ z[y*w + x] = 25; }
```

*const (рекомендован)*

const указание: "я буду только читать"

просто указание: "я буду и читать, и писать!"

## многошерчная адресация

```

int main()
{
    int w * h;
    int * p = malloc(sizeof(int)*w*h);
    p[y*w + x];
}

```

*пунктое построение двумерного массива*

Примечательно в том, что мы получим указатель

```

void f( uint w, uint h, int *z )
{
    z[y*w + x] = 25;
}

```

*const (рекомендован)*

const указатель: "я буду только читать"  
просто указатель: "я буду и читать, и писать!"

Автор:

Аннича Аделина,  
М3132.

Вопросы, ошибки и  
предложения нужно!  
писать сюда;  
tg: @entelechyy